Amendments to the Specification:

Beginning on page 5, please replace paragraph [0013] with the following paragraph where additions are underlined and deletions are in strike-out font:

-- Fig. 1 is a block diagram of an implementation for regulating access to a file system device in a manner that may help to minimize unnecessary device access operations and/or unnecessary device activation-deactivation operations, and thus may help to conserve battery power in a mobile computing platform. As shown in Fig. 1, file system requests 100 ordinarily handled by a file system driver (FSD) 104 may be intercepted and handled in the first instance by an intermediate FSD 102, which may be implemented as a software component running in conjunction with the operating system running on a processor 103. Alternatively, the functionality of the intermediate FSD could be implemented in a variety of ways – e.g., as a native functionality of the operating system. Generally speaking, implementation of an intermediate FSD or its functional equivalent is platform specific. Regulating access to a file system device also could be implemented as a native functionality of the storage device itself, for example, by including appropriate software, firmware and/or hardware in a storage device such as a hard-drive.--

Beginning on page 7, please replace paragraphs [0015]-[0016] with the following paragraphs where additions are underlined and deletions are in strike-out font:

-- When such a predetermined condition is detected, the buffered write requests may be read from physical memory 110 (shown in more detail in 110a) and written to the appropriate device 106. In some situations, the buffer may include two or more write requests 120 and 121 that seek to modify the exact same memory locations 106b in the storage device 106. For example, a first write request 120 may seek to modify a storage location XX 106b to hold the data value 100 and a second, subsequent (i.e., later in time) write request 121 may seek to modify that exact same memory location XX 106b to hold

the data value 101. In this example, buffering and committing all of the write requests in the order received would produce correct results but would be inefficient in that location XX first would be modified to hold 100 but then would be overwritten almost immediately to hold 101.

Accordingly, buffering write requests can be performed intelligently, for example, by deleting 102a an earlier write request 120 from the buffer when a subsequent write request 121 to the same storage location is intercepted and buffered. As a result, the size of the buffer remains smaller than it otherwise would if all write requests to identical storage locations were buffered. Moreover, committing the buffered write requests can be performed more quickly and efficiently since potentially fewer write requests may need to be performed.--

On page 8, please replace paragraph [0017] with the following paragraph where additions are underlined and deletions are in strike-out font:

-- Selectively buffering write requests can be performed on a process-specific or case-by-case basis. For example, certain applications (e.g., application programs such as Microsoft Word or Microsoft PowerPoint) 105 could selectively register with the intermediate FSD 102 (e.g., by reference to a specific application program interface (API) call associated with the intermediate FSD 102) to identify themselves as participating in the selective buffering scheme. Such applications could register, for example, such that only select file types are subject to selective buffering of write requests or all write requests from that application type are subject to selective buffering.

Beginning on page 13, please replace paragraph [0027] with the following paragraph where additions are underlined and deletions are in strike-out font:

-- Next, the process 200 determines whether one or more other conditions may exist that require the device to be accessed to fulfill the write request (212). As noted previously, examples of such conditions may include detecting that the write buffer has

become full, that a certain amount of time has passed, that the battery power is approaching or has reached a specified threshold level, that the computer system is being turned off or put in a standby state, and/or that a user, a process or the operating system has explicitly requested (220) that the write buffer contents be committed to non-volatile storage. Committing the buffer contents to non-volatile storage in this manner is not limited to being performed only as a part of the process 200 but rather may be performed whenever one or more predetermined conditions are detected.--

Beginning on page 19, please replace paragraph [0037] with the following paragraph where additions are underlined and deletions are in strike-out font:

-- For example, assume that the intercepted read request (402) is a request to read a row from a table in a 300 MegaByte (MB) relational database file. In that case, the entire 300 MB file generally is too large to be read entirely and stored in a typical computer system's physical memory. Instead, the process 400 may read in not only the requested row from the database file, but also an additional portion of the database file that is likely to be accessed by future read requests, in block 418. The amount of the file to be read into physical memory may depend on one or more factors, including available physical memory space, user defined parameters, file size and the like, as may be determined by a component making intelligent decisions ~~various rules~~, in block 418A. For instance, the ~~The~~ additional portions of the file to be read into physical memory may be chosen intelligently, for example, using predictive caching techniques or the like. In the database file example, the additional portions to be read (i.e., the superset of the requested file portion) may include adjacent rows, the entire table, and/or other logically related information fields. In this regard, a file access monitor process could be implemented that keeps track of, and identifies trends relating to, which files, and which portions of files, have been accessed recently, e.g., for the last 3-4 days or so. The intelligent decisions as to which file portions to be read into physical memory by process 400 could be based on the information collected and maintained by the file access monitor, in block 418A. In some cases, reading the superset of the requested file portion

<u>into memory requires translating the received read request for the file portion into a plurality of read requests (418B) that collectively cause the superset to be read from the device.</u> Alternatively, or in addition, a secondary memory paging manager (discussed below) could use information from the file access monitor to make intelligent decisions about which page or pages are to be removed from physical memory, e.g., in order to accommodate additional portions of a file read in response to an intercepted read request.—